IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re Patent Application of: ORELL et al.
Serial No.: 10/035,321
Filed: January 4, 2002

For: STREAMING AND MANAGING COMPLEX MEDIA CONTENT ON WEB SERVERS

Group Art Unit: 2144
Examiner: Greg C. Bengzon

DECLARATION UNDER 37 CFR 1.131

Sir:

We, the undersigned, Zohar Sivan, Dror Orell and Hagai Krupnik, hereby declare as follows:

1) We are the Applicants in U.S. Patent Application No. 10/035,321 (hereinafter "the Application"), and are the inventors of the subject matter described and claimed in claims 1-32 therein.

2) Prior to July 25, 2001, we reduced our invention to practice, as described and claimed in the Application, in Israel, a WTO country. We implemented the invention in the form of software code in the Java programming language. This code ran as part of the HotMedia architecture (Release 3.5) produced by IBM.

3) As evidence of the reduction to practice of the present invention, we attach hereto as Exhibit A our project home page prepared by Dror Orell. The home page provides an overview of our work and states that our

solution was completed and demonstrated. The date of completion, which is blacked out in Exhibit A, is prior to July 25, 2001. As stated on page 1 of this exhibit, our invention was found to work for its intended purpose: "Work was completed... when an end-to-end solution was presented along with a demo..."

4) As further evidence of reduction to practice, we attach hereto as Exhibit B Java software code implementing a Java servlet for delivery of multimedia files, as recited in the claims of the Application. The dates blacked out in Exhibit B are also prior to July 25, 2001.

5) The following table shows the correspondence between the elements of the method claims in the present patent application and elements of the material in the appendices:

Claim 1	**Exhibits**
A method for media streaming, comprising:	Exhibit B, page 1: "This demo allows an audio videi [*sic*] streaming client to receive audiofiles from the Network..."
receiving a request from a client to a server via a network in accordance with a Hypertext Transfer	Exhibit B, page 5: The doGet() method "performs the HTTP GET operation. This method is used when the client sends the request params..." HTTP was (and remains) the standard protocol by which clients

In Re: U.S.S.N. 10/035,321
Group Art Unit 2144
Rule 131 Declaration of Sivan, Orell and Krupnik, cont'd

Protocol (HTTP) to stream a media file of a given type;	may submit requests, including requests for media files, to a Web server, as shown in Figure 1 of Exhibit A.
passing the request to a servlet running in conjunction with the server;	Exhibit A, page 4: "Once the user selects a sound track, a servlet request requesting the required tracks, is composed and deployed." Figure 1 in Exhibit A shows that the servlet request is submitted to the server.
parsing the request using the servlet to identify elements of the media file to be transferred to the client;	Exhibit A, page 4: "The servlet… extracts the requested audio and video tracks and composes its response" to the client. The servlet identifies the desired tracks by parsing the client request using the doGetOrPost() method (Exhibit B, page 6).
streaming the identified elements from the server to the client as a HTTP response.	Exhibit B, page 9: The Handle_seek() method of the servlet "sends the client the content of a MultiMedia file…," using "HttpServletResponse."
Claim 2	
2. A method according to claim 1, wherein parsing the request comprises	One example of a "processing action" carried out by the servlet is to seek and skip to a user-specified point in the multimedia file, as noted in Exhibit B, page 1. The

determining a processing action to be applied to the elements of the media file, and wherein streaming the identified elements comprises applying the processing action to the elements.	processing action is determined by parsing the client request parameters using the doGetOrPost() method (pages 6-7), for example: "else if (RequestType.equals("seek")). The processing action is performed by the Handle_seek() method (pages 9-10).
Claim 3	
3. A method according to claim 2, wherein parsing the request comprises determining a parameter applicable to the processing action, and wherein applying the processing action comprises processing the elements of the media file responsive to the parameter.	The servlet uses the doGetOrPost() method in Exhibit B to determine the point to which the client wishes to seek in the multimedia file (the "parameter applicable to the processing action"), and uses Handle_seek() to process the media file elements in order to stream the multimedia file to the client ("applying the processing action") from the desired point.

In Re: U.S.S.N. 10/035,321
Group Art Unit 2144
Rule 131 Declaration of Sivan, Orell and Krupnik, cont'd

Claim 4	
4. A method according to claim 3, wherein determining the parameter comprises determining a limitation on a media playing capability of the client, and wherein the processing action comprises modifying the identified elements in response to the limitation.	This function is not described explicitly in the Exhibits, but it is implicitly a part of the capability of our system to create and play videos with multiple sound tracks (Exhibit A, page 2), i.e., sound tracks that may be modified and provided in response to limitations of the client media playing capabilities.
Claim 5	
5. A method according to claim 4, wherein determining the limitation comprises identifying a network bandwidth, and wherein modifying the identified elements in response to the	This function is not described explicitly in the Exhibits, but it was a part of the system that we implemented. Note that the demos described on page 2 of Exhibit A are designed for network connections of different bandwidths (60 and 128 kByte/sec).

In Re: U.S.S.N. 10/035,321
Group Art Unit 2144
Rule 131 Declaration of Sivan, Orell and Krupnik, cont'd

limitation comprises altering the elements responsive to the network bandwidth.	
Claim 6	
6. A method according to claim 4, wherein determining the limitation comprises determining a resource level provided by the client, and wherein modifying the identified elements comprises selecting the identified elements responsive to the resource level.	This function is not described explicitly in the Exhibits, but it is implicitly a part of the capability of our system to create and play videos with multiple sound tracks (Exhibit A, page 2), i.e., sound tracks that may be selected depending on the available resource levels of different clients.
Claim 7	
7. A method according to claim 2, wherein applying the processing action comprises transcoding at	Transcoding is performed by the Authoring Component described in Exhibit A, page 4: "The VET was modified to pass the HMAT the compressed video and compressed audio data along with several arrays

least one of the elements of the media file into a desired media format.	indicating how they should be packed as individual MVR Media Bitstream frames."
Claim 8	
8. A method according to claim 1, wherein receiving the request comprises receiving a request for a certain portion of the media file, and wherein parsing the request comprises selecting the elements of the media file to be transferred responsive to the request.	As noted above, the servlet uses the Handle_seek() method (Exhibit B, bottom of page 9) to seek and skip to a user-specified point in the multimedia file: "Sends the client the content of a MultiMedia file starting from a time specified bt [*sic*] the Client." The processing action is determined by parsing the client request parameters using the doGetOrPost() method (pages 6-7). Thus, the user requests the portion of the media file beginning at a specified point, and the servlet selects the part of the media file from the specified point onward to transfer to the client.
Claim 9	
9. A method according to claim 8, wherein the elements of the media file comprise an ordered sequence	A video file, as described in Exhibit A, is necessarily made up of an ordered sequence of frames. Selecting a portion of the video file, as specified above in claim 8, inherently involves selecting a

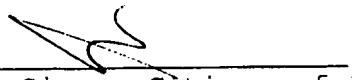
of frames, and wherein selecting the elements comprises selecting a segment within the sequence.	segment of the video frame sequence.
Claim 10	
10. A method according to claim 8, wherein the elements of the media file comprises a plurality of media tracks temporally juxtaposed in parallel, and wherein selecting the elements comprises selecting one or more of the tracks.	Exhibit A, page 2: "The media file consists of multiple tracks and by use of a servlet component we are able to pass over the wire only the tracks that will be played."

6) Claims 11-20 and 22-31 recite apparatus and a computer software product, with limitations similar to those of method claims 1-10. Based on the similarity of subject matter between the method, apparatus and software claims, it can similarly be demonstrated that we reduced to practice the entire invention recited in claims 11-32

prior to July 24, 2001. (Claim 21 adds that the HTTP request and servlet are handled by different servers in a cluster, as is shown in Figure 1 of Exhibit A. Claim 32 adds that the servlet is written in "a platform-independent, object-oriented computer language," such as Java, as in Exhibit B.)

7) Thus, to summarize, the facts set forth above and supported by Exhibits A and B demonstrate that prior to July 25, 2001, we reduced to practice the invention recited in the claims of the Application in the HotMedia demo system that we created, and this system was tested and found to work for its intended purpose.

We hereby declare that all statements made herein of our own knowledge are true and that all statements made on information and conjecture are thought to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application of any patent issued thereon.



Zohar Sivan, Citizen of Israel
13 Odem Street, Zichron Yaakov, Israel

18/6/06
Date

Dror Orell, Citizen of Israel
42a Givat Downs Street, Haifa 34349, Israel

Date

Hagai Krupnik, Citizen of Israel
P.O. Box 225, Nofit 36001, Israel

Date

prior to July 24, 2001. (Claim 21 adds that the HTTP request and servlet are handled by different servers in a cluster, as is shown in Figure 1 of Exhibit A. Claim 32 adds that the servlet is written in "a platform-independent, object-oriented computer language," such as Java, as in Exhibit B.)

7) Thus, to summarize, the facts set forth above and supported by Exhibits A and B demonstrate that prior to July 25, 2001, we reduced to practice the invention recited in the claims of the Application in the HotMedia demo system that we created, and this system was tested and found to work for its intended purpose.

We hereby declare that all statements made herein of our own knowledge are true and that all statements made on information and conjecture are thought to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application of any patent issued thereon.

_____ _____
Zohar Sivan, Citizen of Israel Date
13 Odem Street, Zichron Yaakov, Israel

_____ 06/26/2006
Dror Orell, Citizen of Israel Date
42a Givat Downs Street, Haifa 34349, Israel

_____ _____
Hagai Krupnik, Citizen of Israel Date
P.O. Box 225, Nofit 36001, Israel

prior to July 24, 2001. (Claim 21 adds that the HTTP request and servlet are handled by different servers in a cluster, as is shown in Figure 1 of Exhibit A. Claim 32 adds that the servlet is written in "a platform-independent, object-oriented computer language," such as Java, as in Exhibit B.)

7) Thus, to summarize, the facts set forth above and supported by Exhibits A and B demonstrate that prior to July 25, 2001, we reduced to practice the invention recited in the claims of the Application in the HotMedia demo system that we created, and this system was tested and found to work for its intended purpose.

We hereby declare that all statements made herein of our own knowledge are true and that all statements made on information and conjecture are thought to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application of any patent issued thereon.

_____ _____
Zohar Sivan, Citizen of Israel Date
13 Odem Street, Zichron Yaakov, Israel

_____ _____
Dror Orell, Citizen of Israel Date
42a Givat Downs Street, Haifa 34349, Israel

_____ June 21 2006
Hagai Krupnik, Citizen of Israel Date
P.O. Box 225, Nofit 36001, Israel

EXHIBIT A

Multiple Track Video

IBM Internal Project Home Page
Last update: Oct 25th, 2001
by Dror Orell

Project personnel: Dror Orell

Table of Contents:

Project Status:

Work was completed on [redacted] when an end-to-end solution was presented along with a demo that features dynamic customization of multi-track MVR files.

Work in this field is continued in the context of MPEG4 media customization.

General Overview:

This page describes the activity in the Audiovideo Group of the Haifa Research Lab to create and display video files with multiple audio tracks. With MPEG4 in our mind we aim to create an end to end solution capable of supporting 'object

based' rich video experiences. In the long run we intend to support multiple video objects that the user can interact with, yet our first goal was to create and play videos with multiple sound tracks. The media file consists of multiple tracks and by use of a servlet component we are able to pass over the wire only the tracks that will be played. Only a single video track and a single audio track are streamed to the player, the rest of the audio tracks are filtered by the servlet.

We have developed a full solution: an authoring tool, a java video player and servlet support. With no official MPEG4 streaming format, we have chosen to use the MVR file format and HotMedia architecture as the platform on which to develop this new technology. We have begun working with HotMedia release 3.0 and during our work, when HotMedia3.5 was released had switched to the new classes and authoring tool.

This site includes a chapter detailing a scenario of how the system is used. The scenario outlines the role of all the system's components and the interaction between them. Later on this site includes technical information on each of the components

back to table of contents

Demos:

Two demonstration clips are available:

- Our Main demo page features a clip compressed to fit a connection of 60 kByte/sec.
- An additional demo compressed to 128 kByte/sec.

html, class and media files may be found on ppc034 at the following path:
/usr/lpp/internet/server_root/webpages/hotmediaDemos/multtrack_video/work/

back to table of contents

Scenario for using multiple tracks:

The multitrack video system include three main components, and authoring tool, a servlet component and a player. In this chapter we will describe the end to end process. This process can be divided into three stages that are detailed bellow.

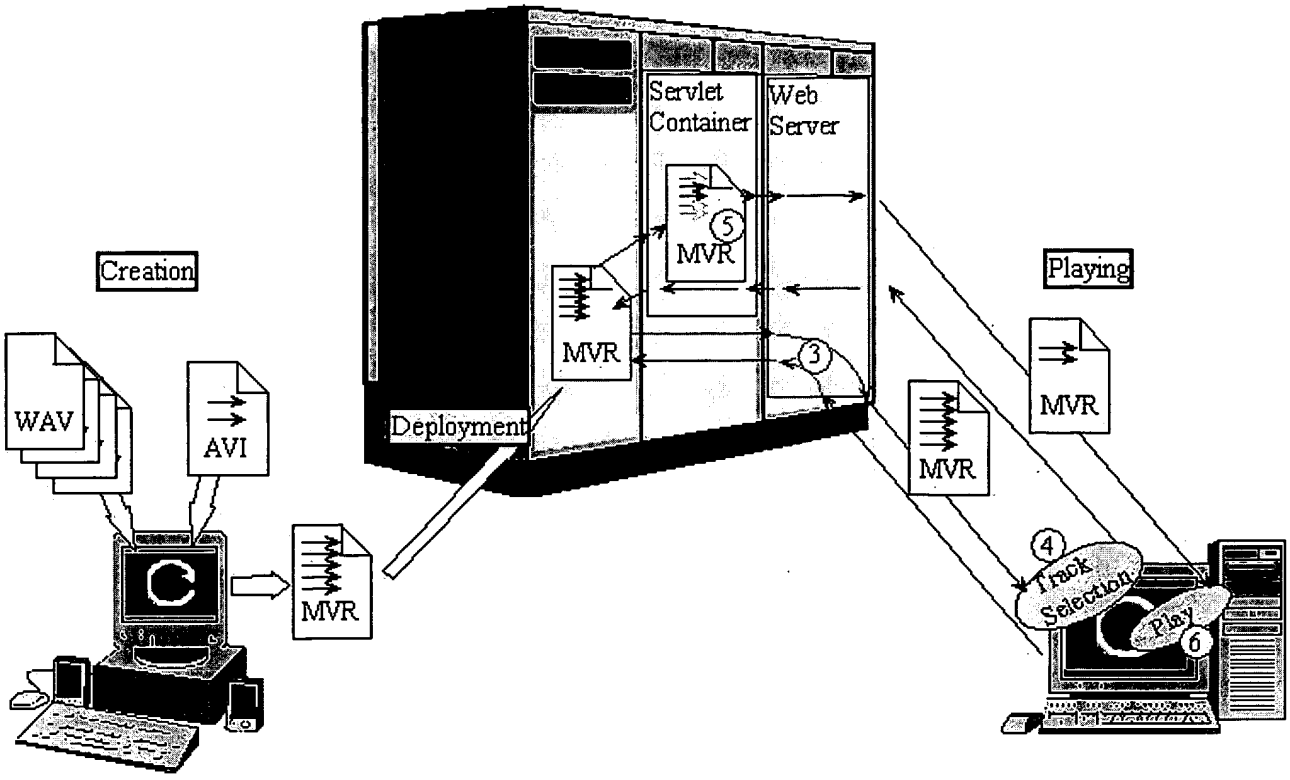


Figure 1 - Scenario of use

Stage 1 - Creation:

A file containing multiple audio tracks is composed from an AVI file and several WAV files. After compressing and AVI the authoring too provides an option to create additional sound tracks by compressing WAV files. The created MVR file will consist of a single video track and multiple audio tracks.

Stage 2 - Deployment:

The MVR file(s) and the Applet classes are mounted on the Web Server. A servlet container is installed and configured to work with the Web Server and the servlet classes are mounted.

Stage 3 - Playing:

1. The user loads to a browser an html page containing the HotMedia Applet.
2. The class files of the video player are loaded to the browser.
3. The MVR file is streamed to the video player and information about the existing audio tracks is recorded. If the file only contains a single audio and single video track then the playing begins, yet if the file contains

multiple audio tracks the user is asked to select the desired sound track and the streaming of the original file is terminated.

4. Once the user selects a sound track, a servlet request requesting the required tracks, is composed and deployed.
5. The servlet, having access to the MVR file, extracts the requested audio and video tracks and composes it's response. The response includes the requested tracks and consists of valid MVR frames.
6. The player plays the tracks received from the servlet as if there were a regular MVR file.

back to table of contents

Authoring Component:

The authoring of MVR video files is handled by two components: the general HotMedia Authoring Tool (*HMAT*) and the plugable dll of the Video Encoding Tool (*VET*). The interface between the HMAT and VET as defined in the interface document supports the passing of multiple tracks, however such functionality was not yet implemented in the HMAT of release 3.0. We've enhanced the VET first to pack the audio and video data in different MVR tracks and then to enable multiple audio tracks. Furthermore, we've defined the enhancements that were required in the HMAT and participated in their implementation.

The VET's original functionality was to compress audio and video data taken from an AVI file, pack them to the HAV format and hand the HAV file to the HMAT. The HMAT would then create an MVR file consisting of a single MVR frame that contained the whole HAV file. The VET was modified to pass the HMAT the compressed video and compressed audio data along with several arrays indicating how they should packed as individual MVR Media Bitstream frames. The VET uses the same multiplexing algorithm that was used in the HAV format, and in fact video and audio parts of each Macro Frame (see the HAV format for more details) are now packed in separate MVR frames. The VET scheduler algorithm ensures that the audio and video tracks are synchronized. Technical details about the code changes in the VET can be found in the following LWP document.

The HMAT is responsible to create a valid MVR file from data passed to it by the VET. Since the handling of multiple tracks had not yet matured we've made several minor adjustments to allow the file creation. Technical details about the code changes in the VET can be found in the following LWP document.

back to table of contents

Player Component:

The HotMedia Applet is designed in a modular way, it consists of a general component and multiple media player. The general component is always loaded to the browser while for each MVR file only the required media players are loaded. The general component is responsible for the parsing of the MVR file and

provides utilities such as GUI and messaging. In order to enable the playing of multi-tracked video we've introduced several fundamental changes in the HotMedia30 Video Player and several modifications in the general architecture. This section will outline the player's structure, for more technical details please refer to the multitrack player design document.

The model used by HotMedia30 considers each media player as an isolated entity interacting only with the *master* class of the general component. Such a model restricts interaction between the different players and in the case of video eliminates the option of using the HotMedia audio player to play the sound track - the synchronization of audio to video requires interaction between the players. The model we've used is different. We've designed the new video player to relay and control the audio player. Therefore, both the audio and the video tracks are sent to the video player. The video player dominates the audio player, it passes it the audio data, controls it's playing (start, stop, pause and mute) and sends time queries that are required for synchronization.

Main changes made in the video player:

- An instance of hm30g723 is used as the audio player. As a result, the internal audio decoder class (hm30avmrtcdec) was removed, the hm30avplayer class no longer needs to extend inputStream and was merged with hm30video.
- The cache holds only compressed video data, MVR frames with audio data are passed to the audio player. Intra frames are marked at the time of loading to the cache.
- All control methods were modified to also pass the action to the audio player.

Changes made in the hm35 class:

- Information about the location of the servlet is extracted from the APPLET tag in the html file and is made available to the hm35 master class.

Changes made in the hm35master class:

- Media frames of audio data, if part of a video file, are directed to the video player.
- A servlet interaction method was introduced - postServletRequest(..).
- Supporting the symbolic track name feature of the MVR file format and passing the symbolic names to the players.
- Minor changes were made in the parsing of the MVR file to support accepting servlet responses (the servlet response does not include the MVR header frame).

Changes made in the audio player:

- A "video mode" was introduced to identify cases when playing under the video player. In such cases all calls to the gui class were suppressed.

Changes made in the hm35player:

- A method was added to allow the video player to interact with the audio player while only keeping a reference to the base player class (this is required to have the audio player class loaded only when audio is present in the stream).

back to table of contents

Servlet Component:

The servlet component is an enhancement of the servlet package for HotMedia40. A track filtering module was added on top to the range (seek) module that was implemented for HotMedia. Requests for either action or both actions are now handled by the servlet component.

The servlet module is used by the multi-track video player when a clip consisting of multiple audio tracks is player. If multiple tracks are detected in the MVR stream, the player freezes and displays the options to the user to choose from. Once the user chooses an audio track, the player composes a servlet request for the chosen track and players the stream received as the servlet result. Event after the servlet response is handled, the user still holds the option to choose another audio track. If the user selects a different track while playing a combined servlet request will be composed, requesting both track filtering and seek action. The response of such a request will start playing from the point in the clip where the user initiated the change.

back to table of contents

Contact:

Dror Orell

IBM Haifa Research Lab.

Audio and Video Technologies Group.

Tel: +972-4-8296-251

LN: Dror Orell/Haifa/IBM@IBMIL

e-mail: orell@il.ibm.com

EXHIBIT B

```
/************************************************************
 *
 * Copyright:      (c) Copyright IBM Corp. [redacted]
 *               Haifa Research Laboratory
 *               Audio/Video Technology Group
 *          All rights reserved.
 *
 *-----------------------------------------------------------
 *
 * Project:      IMG Codecs
 *
 *-----------------------------------------------------------
 *
 * File Name:    HotMediaServlet.java
 *
 * Purpose: Contains the server(Servlet) Java code for HotMedia
project(s).
 *
 * Author:       Hagai Krupnik
 *
 * Email:        hagai@haifa.vnet.ibm.com
 *
 * Date:         [redacted]
 *
 * Remarks:      Java Servlet demo serving multimedia files with seek
caoeabilities.
 ************************************************************/

package com.ibm.hotmedia.hotmediaservlet;
import java.net.*;
import java.lang.*;
import java.lang.reflect.*;
import java.io.*;
import java.util.*;
import javax.servlet.*;                    // servlets
import javax.servlet.http.*;


/**
 * The ImgServlet class is a part of a MutiMedia streaming demo for
the IMG.
 * This demo allows an audio videi streaming client to receive
audiofiles from the Network and <em> seek </em>
 * to another point within the MM file without the need to send all
the skipped data over the net.
 *
 * How to use this servlet is a source code :
 *
 * @see  mmRandomAccessFile
 * @version     2.0 - [redacted]
 * @author  Hagai Krupnik IBM HRL */

  public class HotMediaServlet extends HttpServlet {

  /** Copyright.
   *  <BR><FONT COLOR="#FF0000">
   *  Licensed Materials - Property of IBM
<br>
```

```
   *  "Restricted Materials of IBM"
<br>
   *  5746-SM2
<br>
   *  (c) Copyright IBM Corporation [redacted] All Rights Reserved
<br>
   *  US Government Users Restricted Rights - Use, duplication or
   *  disclosure restricted by GSA ADP Schedule Contract with
   *  IBM Corporation.</FONT>
   **/

  public  static final String a_copyright_notice="(c) Copyright IBM
Corporation [redacted].";
  public  static final String IMG_SERVLET_VERSION="2.0 - [redacted]
[redacted]";
  private static final int ReadWriteChunkSize = 65536;
  /**
   * Path to all the media files
   */
  private  String content_path = null;


  /**
   *counts the number of requests hits since the last
<code>init</code> call.
   */
  private int hits_counter;


  /**
   *  Determine if any POST request will write a line into the
servlet-log.{date} file.
   *  This parameter is initiated during <code>init</code> using data
from <code>/etc/servlet.conf</code> file
   *  (Servlet init parameter). <code>log=true</code> enables
loggings, <code>log=false</code>
   *  disable loggings. Note that in both cases the servlet internal
errors are logged into
   *  the servet-log.{date} file. However, <code>log=false</code>
eliminates the loggings of
   *        successful connections.
   */
  private boolean to_log=false;

  /** Uses the regular httpServlet init but adds the following to it
:
      <OL>
      <LI> Init of hits counters.
      <LI> Init of <code> to_log </code> variable  which determines
if the logging option is  on or off.
      </OL>
   */
  private static Class carray1[]=null;
  private static Class carray2[]=null;

  public void init(ServletConfig config) throws ServletException{
    try {
      super.init(config);
    }
    catch (ServletException e){
         log("super.init(config) - error " +e.toString());
```

```
         throw (e);
        }
    hits_counter = 0;
    log("IBM HotMediaServlet - Init : version " +
IMG_SERVLET_VERSION);
    content_path=getInitParameter("content_path");
    String s = getInitParameter("log");
    if (s!=null)   {
      if (s.equals("true")) to_log = true;
      }
    else to_log = false;
    try {
          carray1 = new Class[1];
          carray1[0]= Class.forName("java.lang.String");
          carray2 = new Class[2];
          carray2[0]= Class.forName("java.lang.String");
          carray2[1]= Class.forName("java.lang.String");
        }
     catch (Exception e) {
      log("Error in Class.forName of String");
     }

  }



  /**
   *Information about this servlet.
   */
  public String getServletInfo()    {
    return "IBM HotMediaServlet version: " + IMG_SERVLET_VERSION
+"\n\n"+
           "streams and seek MultiMedia files created at IBM HRL ";
  }


private boolean printServletInfo(HttpServletRequest
req,HttpServletResponse res)
                       throws ServletException, IOException {

      res.setContentType("text/html");
    ServletOutputStream out = res.getOutputStream();

      out.println("<html>");
      out.println("<head><title>IBM HotMediaServlet - info
page</title></head>");
      out.println("<body>");
/*
      out.println("<h1>Requested URL:</h1>");
      out.println("<pre>");
      out.println (HttpUtils.getRequestURL (req).toString ());
      out.println("</pre>");
*/

    out.println("<center><h1>HotMediaServlet Remote Info
Page</h1></center>");
//    out.println(getServletInfo());
    out.println("<B>description</B> : " + "Streams MultiMedia files
with seek capabilities");
    out.println("<br>");
```

```
    out.println("<B>version</B>      : " + IMG_SERVLET_VERSION);
/*
    out.println("<H2>server info</H2>");
    out.println(getServletContext().getServerInfo());
*/
    Enumeration enum = getServletConfig().getInitParameterNames();
        if (enum != null) {
            boolean first = true;
          while (enum.hasMoreElements()) {
            if (first) {
                  out.println("<h1>Init Parameters</h1>");
                  out.println("<pre>");
                first = false;
                }
            String param = (String) enum.nextElement();
                out.println(" "+param+": "+getInitParameter(param));
          }
          out.println("</pre>");
        }



    //out.println("<B>Server name</B> : " + req.getServerName());
//    out.println("<B>Server port</B> : "  + req.getServerPort());

/*
      out.println("<h1>Request information:</h1>");
      out.println("<pre>");

      print(out, "Request method", req.getMethod());
      print(out, "Request URI", req.getRequestURI());
      print(out, "Request protocol", req.getProtocol());
      print(out, "Servlet path", req.getServletPath());
      print(out, "Path info", req.getPathInfo());
      print(out, "Path translated", req.getPathTranslated());
      print(out, "Query string", req.getQueryString());
      print(out, "Content length", req.getContentLength());
      print(out, "Content type", req.getContentType());
      print(out, "Server name", req.getServerName());
      print(out, "Server port", req.getServerPort());
      print(out, "Remote user", req.getRemoteUser());
      print(out, "Remote address", req.getRemoteAddr());
      print(out, "Remote host", req.getRemoteHost());
      print(out, "Authorization scheme", req.getAuthType());

      out.println("</pre>");

      Enumeration e = req.getHeaderNames();
      if (e.hasMoreElements()) {
          out.println("<h1>Request headers:</h1>");
          out.println("<pre>");
          while (e.hasMoreElements()) {
            String name = (String)e.nextElement();
            out.println(" " + name + ": " + req.getHeader(name));
          }
          out.println("</pre>");
      }

      e = req.getParameterNames();
      if (e.hasMoreElements()) {
```

```
          out.println("<h1>Servlet parameters (Single Value
style):</h1>");
          out.println("<pre>");
          while (e.hasMoreElements()) {
            String name = (String)e.nextElement();
            out.println(" " + name + " = " + req.getParameter(name));
          }
          out.println("</pre>");
      }

      e = req.getParameterNames();
      if (e.hasMoreElements()) {
          out.println("<h1>Servlet parameters (Multiple Value
style):</h1>");
          out.println("<pre>");
          while (e.hasMoreElements()) {
            String name = (String)e.nextElement();
            String vals[] = (String []) req.getParameterValues(name);
            if (vals != null) {
                out.print("<b> " + name + " = </b>");
                out.println(vals[0]);
                for (int i = 1; i<vals.length; i++)
                  out.println("           " + vals[i]);
            }
            out.println("<p>");
          }
          out.println("</pre>");
      }

*/

      out.println("</body></html>");
    return true;
   }


      /**
     * Performs the HTTP GET operation.
     * This method is used when the client sends the request params
at its command line
     * @param req encapsulates the request to the servlet
     * @param resp encapsulates the response from the servlet
     * @exception ServletException if the request could not be
handled
     * @exception IOException if detected when handling the request
     */


    public void doGet(HttpServletRequest req, HttpServletResponse
res)
                        throws ServletException, IOException {


    if (to_log)
          log("HotMediaServlet doGet");
        String s = req.getQueryString();
        if (s==null)
           printServletInfo(req,res);
        else
         doGetOrPost(req,res,s);
```

```
  }  // END of DoGet



     /**
     * Performs the HTTP POST operation.
     * This method is used when the client sends the request params
using a data input stream.
     * @param req encapsulates the request to the servlet
     * @param resp encapsulates the response from the servlet
     * @exception ServletException if the request could not be
handled
     * @exception IOException if detected when handling the request
     */


    public void doPost(HttpServletRequest req, HttpServletResponse
res)
                        throws ServletException, IOException {


    int cl = req.getContentLength();

    //    The next lines are example that should work if the client
sends the correct MIME type request
    //        However - currently it doesn't work on Netscape WIN95
browser so we bypass the problem
    //    int ImageId =
Integer.parseInt(req.getParameter("ImageId"));
    //    String RequestType = req.getParameter("RequestType");
    //    instead we pare it "manually"

    if (to_log)
          log("HotMediaServlet doPost");
    byte b[] = new byte[cl];
    ServletInputStream in = req.getInputStream();
    in.read(b, 0, b.length);
    String s = new String(b);
    doGetOrPost(req,res,s);
    }  // END of DoPost




/**
Unifies the handling of get or post methods (I.e. params can be
passed either at the URL or with a DataInputStream)
*/
private void doGetOrPost(HttpServletRequest req, HttpServletResponse
res, String s)
                        throws ServletException, IOException {

    String filename = req.getPathTranslated();
    Hashtable h= null;
    if (to_log)
          log("filename = " + filename);
    Class c = null;
    Method m = null;
    try {
```

```
        c = Class.forName("javax.servlet.http.HttpUtils");
        m = c.getDeclaredMethod("parseQueryString",carray1);
        Object args[] = new Object[1];
            args[0] = new String(s);
//        log("parseQueryString with a single string");
            h=(Hashtable)m.invoke(res, args);
        }
    catch (Exception e){
         log("6" + e.toString());
         try  {
             m = c.getDeclaredMethod("parseQueryString",carray2);
                 log("parseQueryString with two strings");
             Object args[] = new Object[2];
             args[0] = new String(s);
             args[1] = new String("UTF8");
                   h=(Hashtable)m.invoke(res, args);
              }
            //     h = HttpUtils.parseQueryString(s,"");
            catch (InvocationTargetException e2){
            log("ParseQueryString Or reflector error " +
e2.toString() + " -> " + e2.getTargetException().toString());
            return;
          }
            catch (Exception e2){
              log("ParseQueryString Or reflector error "
+e2.toString());
            return;
          }

        }

    hits_counter++;
    String RequestType = ((String[])h.get("RequestType"))[0];
    // print information to keep track on the calling clients
    String s1 = req.getRemoteAddr();
    String s2 = InetAddress.getByName(s1).getHostName();

    if (RequestType.equals("getContentLength")) {
        if (to_log)
              log("getContentLength request from " + s2 );
        if (null == filename)
              Handle_getContentLength(h,res);
         else
            Handle_getContentLength(h,res, filename);
     }
    else if (RequestType.equals("seek")) {
      if (to_log)
           log("seek request from " + s2 );
         if (null == filename)
            Handle_seek(h,res);
              else
             Handle_seek(h,res,filename);
      }


    else if (RequestType.equals("getInfo")) printServletInfo
(req,res);
    else if(to_log)
            log ("Unsupported Request of Type " + RequestType + "
from " + s2) ;
```

```
}

/**
* Sends the client the content-length of a MultiMedia file both in
bytes and in mSec.
* @param h  Hashtable with the parsed *resolved) query string of the
request.
* @param res  the HttpResvletResponse to allow opening stream with
the client.
* @exception Exception on any failure to perform the task.
* @return  true on succses.
*/


private boolean Handle_getContentLength (Hashtable h,
HttpServletResponse res) {

     String filename = content_path+((String[])h.get("fileName"))[0];
     return   Handle_getContentLength(h,res,filename);
    }


private boolean Handle_getContentLength (Hashtable h,
HttpServletResponse res, String filename) {

     String fileType = null;
//     String filename =
content_path+((String[])h.get("fileName"))[0];

     try {
        fileType = ((String[])h.get("fileType"))[0];
       }
       catch (Exception e)  {   // just leave the string as null
       }
       long duration = -1;
       long nob = 1;           // to have a positive contentLength
       long BytesInFrame=-1;
       int ErrorId=0;
     int headerSize=0;
     byte FirstBytes[]=null;


     mmRandomAccessFile mmraf =null;


    // FirstBytes[0]= (byte)hits_counter;
    // open file and extract information
    try {
      if (null != fileType)
            mmraf = new  mmRandomAccessFile(filename,fileType);
       else
            mmraf = new  mmRandomAccessFile(filename);

      nob      = mmraf.length();
      duration = mmraf.availableTime();
      BytesInFrame = mmraf.BytesInFrame();

      headerSize = mmraf.getFileHeaderSize();
      if (headerSize ==0) {
//           FirstBytes = new byte[1];
//         FirstBytes[0]=42;
```

```
            FirstBytes = new byte[2];
           FirstBytes[0]=(byte)ErrorId;
            FirstBytes[1]=(byte)BytesInFrame;

         }
      else {
         FirstBytes = new byte[headerSize];
         mmraf.read(FirstBytes,0,headerSize);
         }
      mmraf.close();
      }
      catch (Exception e) {
         if (to_log)
          log("Handle_getContentLength Exception : " +e.toString());
          ErrorId=-1;
      }


    // preper response
       if (nob<=0) ErrorId=-1;
       res.setContentType("application/octet-stream");
       res.setContentLength((int)nob);
       res.setHeader("ContentDuration",String.valueOf(duration));
       res.setHeader("Error",String.valueOf(ErrorId));
       res.setHeader("BytesInFrame",String.valueOf(BytesInFrame));

         try  {
            ServletOutputStream out = res.getOutputStream();
       // This is needed because if we don't send any byte
         // some servers will not send the http header as well.
        if (headerSize ==0) {
              out.write(FirstBytes,0, 2);  // ErrorId + BytesInFrame
//            out.write(FirstBytes,0, 1);
           }
         else
              out.write(FirstBytes,0, headerSize);
           out.flush();
           out.close();
         }
      catch (Exception e) {
        if(to_log)
              log("Exception in getContentLength while sending bytes
to client " + e.toString());
              }

      return(true);
    }


/**
* Sends the client the content of a MultiMedia file starting from a
time
* specified bt the Client.
* @param h Hashtable with the parsed *resolved) query string of the
request.
* @param res the HttpResvletResponse to allow opening stream with the
client.
* @exception Exception on any failure to perform the task.
* @return  true on succses.
*/
```

```
private boolean Handle_seek(Hashtable h, HttpServletResponse res) {

   // parse variables from the hash table
    String filename = content_path+((String[])h.get("fileName"))[0];

    return Handle_seek(h,res,filename);

    }

private boolean Handle_seek(Hashtable h, HttpServletResponse res,
String filename) {


    long seek =0;
    long nob = 1;          // make sure contentLength is possitive
    long duration = -1;
    long startTime = 0;
    int ErrorId =0;
    int bytesLeft =0;
    byte[] DataBytes= null;
    String filetype = null ;
    mmRandomAccessFile mmraf = null;

    // OLD VERSION when filename was part of the query string parse
variables from the hash table
    // filename = content_path+((String[])h.get("fileName"))[0];

    try {
         filetype = ((String[])h.get("fileType"))[0];
       }
    catch (Exception e)  {     // just leave the string as null
       }

    seek =  Long.parseLong(((String[])h.get("offset"))[0]);

    try {
      if (null != filetype)
            mmraf = new  mmRandomAccessFile(filename,filetype);
       else
                  mmraf = new  mmRandomAccessFile(filename);
      nob = mmraf.length();
      duration = mmraf.availableTime();

      if (seek!=0) {
         nob -= mmraf.seekToTime(seek);
         startTime = mmraf.getStartTime();
         }


      // handle "Empty" files
      if (nob<=0) {
      ErrorId=-1;
      nob=1;
      }


      if (to_log)
            log("length=" +mmraf.length() + " nob=" + nob + "
duration=" + duration +" startTime=" + startTime);
      }
      catch (Exception e) {
```

```
      if (to_log)
            log(" nob=" + nob + " duration=" + duration +"
startTime=" + startTime);

      if(to_log)
            log("Seek Exception openning or seeking within file: "
+e.toString());
      ErrorId=-1;
      nob=1;
      }

    DataBytes = new byte[ReadWriteChunkSize];

    bytesLeft = (int) nob;
    try {
      // Set HTTP response header fields
        res.setContentType("application/octet-stream");
        res.setContentLength((int)nob);
        res.setHeader("ContentDuration",String.valueOf(duration-
startTime));
        res.setHeader("StartTime",String.valueOf(startTime));
      res.setHeader("Error",String.valueOf(ErrorId));

      // open output stream

      ServletOutputStream out = res.getOutputStream();
      // read-write loop
      while (bytesLeft>ReadWriteChunkSize) {
        bytesLeft-=ReadWriteChunkSize;
        mmraf.read (DataBytes ,0 ,ReadWriteChunkSize);
          out.write  (DataBytes ,0 ,ReadWriteChunkSize);
        }
      mmraf.read (DataBytes ,0 , bytesLeft);
      out.write  (DataBytes ,0 , bytesLeft);

      // close streams
      mmraf.close();
        out.flush();
        out.close();
     }
      catch (Exception e) {
         if(to_log)
               log("Seek Exception reading from file or writing to
net: " +e.toString());
      }

      return(true);
    }// handle_seek


}    // END HotMediaServlet
```